Taller: **Data Quality Gate - Scripts**

Proyecto Big Data I

VS Code + conda + Python

José Luis Gómez Ortega

2025

1 Intro (motivación)

Imagina que cargas encuestas de turismo (Alcoi) a tu base de datos para un dashboard. Algunos registros vienen con tildes en columnas, enteros como texto, combinaciones imposibles (*0 adultos y 0 menores*), valores ambiguos como "No entiendo la pregunta" y duplicados por marca temporal. Si eso llega a producción, pierdes tiempo y credibilidad.

Un **Data Quality Gate** es un guardia de frontera: **lee** tu dataset, **aplica reglas explícitas**, **genera un informe** y si detecta problemas críticos, **termina con código 1** para *bloquear la ingestión*. Hoy construimos ese gate como un **script** (no notebook): simple, reproducible y listo para enganchar en cualquier pipeline.

2 Objetivos

- Diferenciar **notebook** vs **script** y por qué el script es ideal para checks previos a ETL/ML.
- Ejecutar un script con **argumentos** y comprender el **exit code** (0/1).
- Dominar **rutas** con `pathlib` para evitar "file not found" según desde dónde ejecutes.
- Aplicar reglas de calidad: **esquema, tipos, dominios, rangos, unicidad y condicionales**.
- Generar un **informe CSV** de incidencias y usarlo para decidir *OK/Bloquear*.
- Entender cómo esto evita **DBs contaminadas**, **KPIs erróneos** y **modelos rotos**.

3 Conceptos clave (qué, dónde y por qué)

Script vs Notebook

Qué: un script hace una tarea y termina (no interactivo). **Dónde**: pasos automatizados de pre-ingesta/pre-train. **Por qué**: reproducible, integrable (cron, Airflow, CI).

CLI y argumentos

Qué: invocar `dq_check.py` con `-i`, `-r`, `-o`. **Dónde**: capa de orquestación del check. **Por qué**: configuración sin tocar código.

Rutas con `pathlib`

Qué: construir rutas relativas al archivo (`__file__`) y no al *cwd*. **Dónde**: lectura de `data/`, `config/` y escritura en `out/`. **Por qué**: portabilidad real.

Reglas de calidad

Esquema, **Tipos**, **Dominios**, **Rangos**, **Unicidad**, **Condicionales** (si `origen=España` entonces `ciudad_es` no nula; si `Internacional` entonces `pais_intl` no nulo; `n_adultos + n_menores  1`).

Informe & Exit code

Qué: CSV con incidencias + `exit(0/1)`. **Dónde**: artefacto en `out/`. **Por qué**: trazabilidad y automatización.

4 Flujo completo (todo resuelto y con archivos indicados)

4.1 Estructura mínima del proyecto

Listing 1: Árbol de directorios

```
project/
  data/
    raw/alcoy_sample.csv
  config/
    rules.yml
  out/
  scripts/
    dq_check.py
  requirements.txt
  README.md
```

4.2 Dependencias

Archivo: `requirements.txt`

Listing 2: requirements.txt

```
pandas
pyyaml
```

4.3 Reglas preparadas (ajusta encabezados exactos si difieren)

Archivo: `config/rules.yml`

Listing 3: config/rules.yml

```yaml
columns_map:
  timestamp: "Marca temporal"
  origen: "Cul es su procedencia?"
  ciudad_es: "En el caso de Espaa, qu ciudad?"
  pais_intl: "En el caso de Internacional, qu pas?"
  n_adultos: "# Con cuntas personas viajas? (adultos)"
  n_menores: "# Con cuntos menores viajas?"
  fin_semana: "Con el fin de semana"

schema:
  required: [timestamp, origen, n_adultos, n_menores, fin_semana]

types:
  timestamp: datetime
  origen: category
  ciudad_es: string
  pais_intl: string
  n_adultos: int
  n_menores: int
  fin_semana: category

domains:
  origen:
    allowed: ["Alcoi", "Espaa", "Internacional", "Comunidad Valenciana"]
  fin_semana:
    allowed: ["S", "No", "No entiendo la pregunta"]

ranges:
```

```
  n_adultos: {min: 0, max: 10}
  n_menores: {min: 0, max: 10}

uniques:
  - [timestamp, origen]

conditionals:
  - if: {origen: "Espaa"}
    then: {require_not_null: ["ciudad_es"]}
  - if: {origen: "Internacional"}
    then: {require_not_null: ["pais_intl"]}
  - rule: "n_adultos + n_menores >= 1"

severity:
  missing_column: error
  type_mismatch: error
  domain_violation: warning
  range_violation: warning
  uniqueness_violation: error
  conditional_violation: error
```

4.4 Script principal (súper comentado)

Archivo: `scripts/dq_check.py`

Listing 4: scripts/dq_check.py

```python
#!/usr/bin/env python3
"""
Data Quality Gate simple para CSVs de encuestas (Alcoi).
- Lee CSV y reglas YAML.
- Mapea nombres reales -> cannicos.
- Fuerza tipos.
- Ejecuta checks y guarda informe CSV.
- Exit code: 0 (OK) / 1 (errores crticos).
"""
import sys, argparse, logging
from pathlib import Path
import pandas as pd, yaml

# --- Rutas robustas relativas a este archivo ---
HERE = Path(__file__).resolve().parent     # .../project/scripts
PROJECT_ROOT = HERE.parent                 # .../project
DATA_DIR = PROJECT_ROOT / "data"
CONFIG_DIR = PROJECT_ROOT / "config"
OUT_DIR = PROJECT_ROOT / "out"

# --- Logging y argumentos CLI ---
def setup_logging(level: str = "INFO"):
    logging.basicConfig(
        level=getattr(logging, level.upper(), logging.INFO),
        format="%(asctime)s | %(levelname)s | %(message)s",
    )

def parse_args():
    p = argparse.ArgumentParser(description="Data Quality Gate (simple)")
    p.add_argument("-i","--input", required=True, help="Ruta al CSV de entrada")
    p.add_argument("-r","--rules", default=str(CONFIG_DIR/"rules.yml"),
                   help="Ruta al archivo de reglas YAML")
    p.add_argument("-o","--out", default=str(OUT_DIR),
```

```
                    help="Carpeta para el informe")
    p.add_argument("--log-level", default="INFO")
    return p.parse_args()

# --- Reglas (YAML) ---
def load_rules(path: Path) -> dict:
    with open(path, "r", encoding="utf-8") as f:
        data = yaml.safe_load(f)
    return data or {}

# --- Renombrado y tipos ---
def rename_columns_to_canonical(df: pd.DataFrame, columns_map: dict) -> pd.DataFrame:
    if not columns_map:
        return df
    real_to_canon = {v: k for k, v in columns_map.items()}
    return df.rename(columns=lambda c: real_to_canon.get(c, c))

def coerce_types(df: pd.DataFrame, types_cfg: dict) -> pd.DataFrame:
    if not types_cfg:
        return df.copy()
    out = df.copy()
    for col, t in types_cfg.items():
        if col not in out.columns:  # si la columna no existe, seguimos
            continue
        try:
            if t == "int":
                out[col] = pd.to_numeric(out[col], errors="coerce").astype("Int64")
            elif t == "float":
                out[col] = pd.to_numeric(out[col], errors="coerce")
            elif t == "datetime":
                out[col] = pd.to_datetime(out[col], errors="coerce", dayfirst=True)
            elif t == "category":
                out[col] = out[col].astype("string").astype("category")
            elif t == "string":
                out[col] = out[col].astype("string")
        except Exception as e:
            logging.warning("No se pudo convertir %s a %s: %s", col, t, e)
    return out

# --- Checks: devuelven lista de 'findings' dicts ---
def check_schema(df, rules):
    required = set((rules.get("schema") or {}).get("required", []))
    missing = sorted(list(required - set(df.columns)))
    findings = []
    for col in missing:
        findings.append({"rule":"missing_column","column":col,"row":None,
                         "detail":"Columna obligatoria ausente"})
    return findings

def check_domains(df, rules):
    findings = []
    for col, cfg in (rules.get("domains") or {}).items():
        if col not in df.columns:
            continue
        allowed = set(cfg.get("allowed", []))
        mask = (~df[col].isna()) & (~df[col].isin(allowed))
        for idx in df[mask].index:
            findings.append({"rule":"domain_violation","column":col,"row":int(idx),
                             "detail":f"Valor={df.at[idx,col]!r} no permitido"})
    return findings
```

```
94
95  def check_ranges(df, rules):
96      findings = []
97      for col, cfg in (rules.get("ranges") or {}).items():
98          if col not in df.columns:
99              continue
100         mn, mx = cfg.get("min"), cfg.get("max")
101         if mn is not None:
102             bad = (df[col] < mn)
103             for idx in df[bad.fillna(False)].index:
104                 findings.append({"rule":"range_violation","column":col,"row":int(idx),
105                                  "detail":f"< min({mn})"})
106         if mx is not None:
107             bad = (df[col] > mx)
108             for idx in df[bad.fillna(False)].index:
109                 findings.append({"rule":"range_violation","column":col,"row":int(idx),
110                                  "detail":f"> max({mx})"})
111     return findings
112
113 def check_uniques(df, rules):
114     findings = []
115     for cols in (rules.get("uniques") or []):
116         dups = df.duplicated(subset=cols, keep=False)
117         for idx in df[dups].index:
118             findings.append({"rule":"uniqueness_violation",
119                              "column":",".join(cols),"row":int(idx),
120                              "detail":"Fila duplicada para la clave dada"})
121     return findings
122
123 def check_conditionals(df, rules):
124     findings = []
125     for cond in (rules.get("conditionals") or []):
126         if "rule" in cond:
127             expr = cond["rule"]                    # p.ej. "n_adultos + n_menores >= 1"
128             bad = ~df.eval(expr)                   # incumplimientos
129             for idx in df[bad.fillna(True)].index:
130                 findings.append({"rule":"conditional_violation","column":None,
131                                  "row":int(idx),"detail":f"No cumple: {expr}"})
132         else:
133             if_ = cond.get("if", {})
134             then = cond.get("then", {})
135             mask = pd.Series(True, index=df.index)
136             for c, v in if_.items():
137                 mask = mask & (df[c] == v)
138             req = then.get("require_not_null", []) or []
139             for col in req:
140                 bad = mask & df[col].isna()
141                 for idx in df[bad].index:
142                     findings.append({"rule":"conditional_violation","column":col,
143                                      "row":int(idx),"detail":f"{col} requerido cuando {if_}"})
144     return findings
145
146 def apply_severity(findings, rules):
147     sev_map = (rules.get("severity") or {})
148     for f in findings:
149         f["severity"] = sev_map.get(f["rule"], "warning")
150
151 # --- Programa principal ---
152 def main():
153     args = parse_args()
```

```
    setup_logging(args.log_level)

    out_dir = Path(args.out)
    out_dir.mkdir(parents=True, exist_ok=True)

    logging.info("Leyendo CSV: %s", args.input)
    df_raw = pd.read_csv(args.input)  # ajusta sep/encoding si tu CSV lo requiere
    rules = load_rules(Path(args.rules))

    df = rename_columns_to_canonical(df_raw, rules.get("columns_map"))
    df = coerce_types(df, rules.get("types"))

    findings = []
    findings += check_schema(df, rules)
    findings += check_domains(df, rules)
    findings += check_ranges(df, rules)
    findings += check_uniques(df, rules)
    findings += check_conditionals(df, rules)

    apply_severity(findings, rules)
    summary = {}
    for f in findings:
        summary[f["severity"]] = summary.get(f["severity"], 0) + 1
    logging.info("Incidencias por severidad: %s", summary or "sin incidencias")

    report_path = out_dir / "dq_report.csv"
    pd.DataFrame(findings).to_csv(report_path, index=False)
    logging.info("Informe: %s", report_path)

    has_errors = any(f["severity"] == "error" for f in findings)
    if has_errors:
        logging.error("Errores crticos detectados. Abortando (exit=1).")
        sys.exit(1)
    else:
        logging.info("Calidad OK. (exit=0)")
        sys.exit(0)

if __name__ == "__main__":
    main()
```

4.5 Ejecución base (con salida y exit code)

Terminal (Linux/Mac):

Listing 5: Comando base

```
python scripts/dq_check.py -i data/raw/alcoy_sample.csv -r config/rules.yml -o out
echo $?     # muestra 0 si OK, 1 si errores crticos
```

PowerShell (Windows):

```
python scripts/dq_check.py -i data\raw\alcoy_sample.csv -r config\rules.yml -o out
$LASTEXITCODE
```

Resultado esperado: se genera `out/dq_report.csv`. En consola se imprime un resumen por severidad. Si hay severidad `error`, el exit code es 1 (bloquearías la carga a BBDD).

4.6 Forzar un fallo y entender la política de severidad

Archivo: `config/rules.yml`
Cambia temporalmente el rango para provocar incidencias y observar el *exit code*:

```
ranges:
  n_adultos: {min: 5, max: 5}   # absurdo a propsito (slo 5)
```

Ejecuta de nuevo y mira `out/dq_report.csv`. Si sólo aparecen `warning`, eleva la severidad:

```
severity:
  range_violation: error
```

Vuelve a ejecutar: ahora el *exit code* debe ser `1`.

4.7 Condicionales típicas del dataset de Alcoi

Ya incluidas en `config/rules.yml`:

- Si `origen=España` ⇒ `ciudad_es` no nula.
- Si `origen=Internacional` ⇒ `pais_intl` no nulo.
- `n_adultos + n_menores  1`.

Edita 1–2 filas del CSV para simular incumplimientos y comprueba que aparecen como `conditional_violation` (por defecto, `error`).

4.8 Duplicados (unicidad)

En `config/rules.yml`:

```
uniques:
  - [timestamp, origen]
```

Duplica una fila con la misma pareja y ejecuta: verás `uniqueness_violation` y *exit* = 1.

4.9 Gotchas típicos & soluciones

- **Rutas**: construimos con `Path(__file__)`; no depende del *cwd*.
- **Encoding/Separador**: si tu CSV es *latin-1* o usa ';', edita en `scripts/dq_check.py`, función `main()`, línea de lectura:

```
df_raw = pd.read_csv(args.input, sep=";", encoding="latin-1")
```

- **Tildes y espacios en encabezados**: por eso mapeamos a nombres canónicos (`config/rules.yml` → `columns_map`).

5 Guía exprés para el/la docente

1. Prepara el árbol de proyecto y coloca tu CSV real en `data/raw/alcoy_sample.csv`.
2. Ajusta `config/rules.yml` → `columns_map` a los encabezados *exactos*.
3. Instala dependencias: `pip install -r requirements.txt`.
4. Ejecuta el script y abre `out/dq_report.csv` en clase.

5. Muestra cómo cambiar **severidades** para forzar *exit* = 1.

6. Conecta con casos reales: *"esto corre antes de* `to_sql()`*"*.

6 FAQ

No encuentra `rules.yml`: ejecuta desde la raíz del proyecto.
UnicodeDecodeError: añade `encoding="latin-1"` en `pd.read_csv`.
ParserError: añade `sep=";"` en `pd.read_csv`.
No salen incidencias: revisa `columns_map` (debe coincidir *exacto* con los encabezados del CSV).

Un script sencillo hoy = horas de limpieza ahorradas mañana.
Data Quality Gate antes de cualquier ETL/ML/BI. #shipit